Georgia Institute of Technology

Schools of Computer Science and Electrical & Computer Engineering

CS 4290/6290, ECE 4100/6100: Spring 2016 (Conte)

Project 2: Tomasulo Algorithm Pipelined Processor

Due Dates:

Checkpoint 1: Before 11:55 PM on March 31 2016

Checkpoint 2: Before 11:55 PM on April 8 2016

Checkpoint 3: Before 11:55 PM on April 15 2016

VERSION: 5 (See changelog at end)

Rules

The rules for project 2 are the same as project 1:

1. All students (CS 4290/6290, ECE 4100/6100) must work *alone*.
2. Sharing of code between students is viewed as cheating and will receive appropriate action in accordance with University policy.
3. It is acceptable for you to compare your results with other students to help debug your program. It is not acceptable to collaborate on the simulator design or the final experiments.
4. You should do all your work in the C or C++ programming language, and should be written according to the C99 or C++11 standards, using only the standard libraries.
5. The project may be updated if errors are discovered. It is your responsibility to check the website often and download new versions of this project description as they become available.
6. A Makefile with the frontend will be given to you; you will only need to fill in the empty functions and any additional subroutines you will be using. You will also need to fill in the statistics structure that will be used to output the results.

2. Checkpoint 2

For checkpoint 2, you will enhance your validated code from checkpoint 1. The framework for checkpoint 2 is backward compatible with the framework for checkpoint 1. In other words, your checkpoint 1 code *should* run identically on both frameworks.

The only difference is that the 2nd framework supports additional parameters and statistics.

For brevity, only the enhancements to checkpoint 1 will be detailed. You may assume that all specifications from checkpoint 1 are still valid for checkpoint 2, unless noted below.

2.1. Command-Line Parameters

Your project should include a Makefile, which builds binary in your project's root directory named *procsim*. The program should run from this root directory as:

./procsim –r R –f F –j J –k K –l L **–e E –s S** < trace_file

The command line parameters are as follows:

- R – Result buses
- F – Fetch rate (instructions per cycle)
- J – Number of k0 function units
- K – Number of k1 function units
- L – Number of k2 function units
- **E – Exception rate (every E ~~cycles~~ instructions) (NEW)**
- **S – The repair scheme to handle exceptions (see below) (NEW)**
- trace_file – Path name to the trace file

2.2. Exception Rate

- An exception rate of E means every Eth instruction causes an exception. For example, with the default E of 250, this means instructions 250, 500, 750, …, 100000 will cause exceptions.
- Exception instructions will only trigger once. For example, instruction 250 will trigger an exception the first time it's in state update, but will not trigger an exception when you re-run it.
- **In both schemes,** instructions should be tagged as exceptions when completed and should be handled during state update.

2.3. Repair Schemes

Value of S	Scheme to Use
1	*ROB with Bypass*

2	*Checkpoint Repair*

2.3.1. ROB with Bypass

2.3.1.1. The Re-Order Buffer

- The size of the ROB is the same as the size of the scheduling queue, i.e., 2*(number of k0 function units + number of k1 function units + number of k2 function units)
- The dispatch unit allocates ROB entries in program order.
- ~~If there are multiple independent ROB entries ready to broadcast during the same cycle, service them in program order (i.e., lowest PC value to PC).~~
- In any given cycle, the state update unit can only retire consecutively finished instructions (starting from the ROB head). If instruction X has completed but there is a non-completed instruction closer to the ROB head, instruction X cannot be retired.

2.3.1.2. Clock Propagation

The new order of events is as follows:

Cycle Portion	**Action**
1	Handle exceptions or mark completed instructions at the ROB head as retired
2	Broadcast results on the result bus and mark instruction as completed
3	The **ROB** is written via a result bus and marks exceptions
4	Any independent instruction in the scheduling queue is marked to fire and marked as issued
5	Scheduling queue is updated via a result bus

6	The dispatch unit reserves slots in the scheduling queues
7	The dispatch unit (1) **reads the ROB (bypass) (and then the register file if ROB entry not found), then (2) adds the instruction to the ROB**
8	The state update unit deletes retired instructions from the scheduling queue **and the ROB**
9	Fetch instructions
10	Program counter updated

2.3.1.3. Exception Handling

When an exception occurs:

1. Flush the ROB
2. Flush the entire dispatch queue
3. Flush the entire scheduling queue
4. Reset the entire scoreboard
5. Clear all pending broadcasts

In the next cycle, re-start fetching from the **PC of the ROB head when the exception occurred.** In other words, treat the exception as a fault.

The exception handler has a latency/penalty of **1 cycle**.

2.3.2. Checkpoint Repair

2.3.3.

2.3.3.1. Instruction Barriers

The dispatch unit defines barriers are set as follows:

- Define the first barrier in dispatch at instruction 20.
- When the barrier instruction retires, define the new barrier as the youngest instruction in the ~~dispatch queue~~ scheduling queue at the beginning of that cycle (the instruction with the highest PC).

- Continue this loop.

2.3.3.2. Clock Propagation

Assume the following order of events:

Cycle Portion	Action
1	Handle exceptions or mark completed instructions as retired **and, if barrier reached, copy backup1 to backup2 and mark new IB1**
2	Broadcast results on the result bus and mark instruction as completed
3	**The messy (and possibly backup 1) register files** are written via a result bus
4	Any independent instruction in the scheduling queue is marked to fire
5	Scheduling queues are updated via a result bus and marks exceptions
6	The dispatch unit reserves slots in the scheduling queues ~~**and marks instruction barrier**~~
7	The dispatch unit reads the register file
8	The state update unit deletes retired instructions from the scheduling queue
9	Fetch instructions
10	Program counter updated

2.3.3.3. Exception Handling

When an exception occurs:

1. Copy backup 2 to backup 1 and the messy register file (and all registers are marked as ready).
2. Flush the entire dispatch queue
3. Flush the entire scheduling queue
4. Reset the entire scoreboard
5. Reset all busy register file busy bits
6. Clear all pending broadcasts

In the next cycle, re-start fetching from the **PC of the instruction after IB2.** In other words, treat the exception as a fault.

The exception handler has a latency/penalty of **1 cycle**.

2.4. Output

Same as checkpoint 1, unless noted below.

2.4.1. Log File Details

Same as checkpoint 1, with new additions:

Log File Stage	**Represents the Cycle in Which**
RE-FETCHED	Fetch stage pushes a previously-flushed instruction into dispatch queue.
BROADCASTED	A completed instruction is placed on a result bus.
EXCEPTION	The state update unit detects an exception.
BACKUP2 (CPR only)	All instructions before IB1 have retired and Backup 1 is copied to Backup 2

Again, you do not need to match this log file. This is simply there to help debug your code.

2.4.2. Output File Details

- If an instruction was flushed, the output file shows the corresponding cycles for **the last time** it was in a given stage.

2.4.3. Statistics

The simulator outputs the following statistics after completion of the experimental run:

1. Total number of instructions in the trace
2. Average dispatch queue size
3. Maximum dispatch queue size
4. Average number of instructions fired per cycle
5. Average number of instructions retired per cycle (IPC)
6. Total run time (cycles)
7. **Total number of register file hits (NEW)**
8. **Total number of ROB hits (NEW)**
9. **Total number of exceptions (NEW)**
10. **Total number of backup copies from B1 to B2 (NEW)**
11. **Total number of flushed instructions: any non-retired instruction that was dispatched some time before the exception occurred (NEW)**

Notes:

- Number of backups will be 0 when implementing ROB with bypass.
- Number of ROB hits will be 0 when implementing checkpoint repair.

2.5. Validation

Your simulator **must** match the validation output that we will place on T-Square.

2.6. Submission Instructions for Checkpoint 2

Submit the following files in a single compressed folder:

1. Makefile
2. procsim.cpp
3. procsim.hpp
4. procsim_driver.cpp

- Name the folder p2_c2_<your prism id>.tar.gz, for example, p2_c2_prabbat3.tar.gz.
- Please compress the folder using the tar.gz compression format only.
- Do not include the traces in your submission.

2.7. Grading for Checkpoint 2

Checkpoint 2 is worth a total of 20 points:

- +10% Your simulator matches the checkpoint 2 validation output for ROB w/ bypass.
- +10% Your simulator matches the checkpoint 2 validation output for checkpoint repair.

NOTE: Your checkpoint 2 grade is independent of your checkpoint 1 grade. You can receive points for checkpoint 2 regardless of your checkpoint 1 score.

2.8. Changelog

All changes are marked in red.

3/21/16 10:30AM [Paul]:

- Project description:
  - 2.2.1.2: Updated clock propagation.
  - 2.2.1.3: Register file has no busy bits.
  - 2.2.*.3: Penalty is 1 cycle.

3/21/16 4:30PM [Paul]:

- Project description:
  - 2.2.1.1: Clarified when an instruction can be retired.
  - 2.2.1.3: Be sure to clear pending broadcasts when an exception occurs.
  - 2.3.1: Added log file details.
  - 2.3.2: Added output file details.
- Posted validated output for ROB w/ Bypass (version 1).

3/23/16 12:15PM [Paul]:

- Project description:
  - 2.1: E is instructions, not cycles.
  - 2.2: More detail about exception rate E.
  - 2.3.1.1: Details about when an instruction can retire.
  - 2.3.1.2: Updated clock propagation table.
  - 2.3.2.1: Set the new IB1 to the youngest instruction in the scheduling queue, not the dispatch queue.
  - 2.4.1: New log file components
  - 2.4.2: Output file updates
  - 2.4.3: Clarified the definition of a flushed instruction.
- Validated Output: ROB w/ Bypass Updated (version 2).

3/24/16 4:15PM [Paul]:

- Project Description:
    - 2.3.2.2: Updated propagation table
    - 2.4.1. Added “BACKUP2” to log file.
- Validated Output: ROB w/ Bypass Updated (version 3)
    - Fixed a bug, same implementation
- Validated Output: CPR Posted (version 1)